● ASSIGNMENT - 1: DTB : SUPPLY CHAIN MANAGEMENT

Team Members:
Kshitij Gutpa: 2021201075
Annapoorani Anantharaman: 2021201017

Project structure:

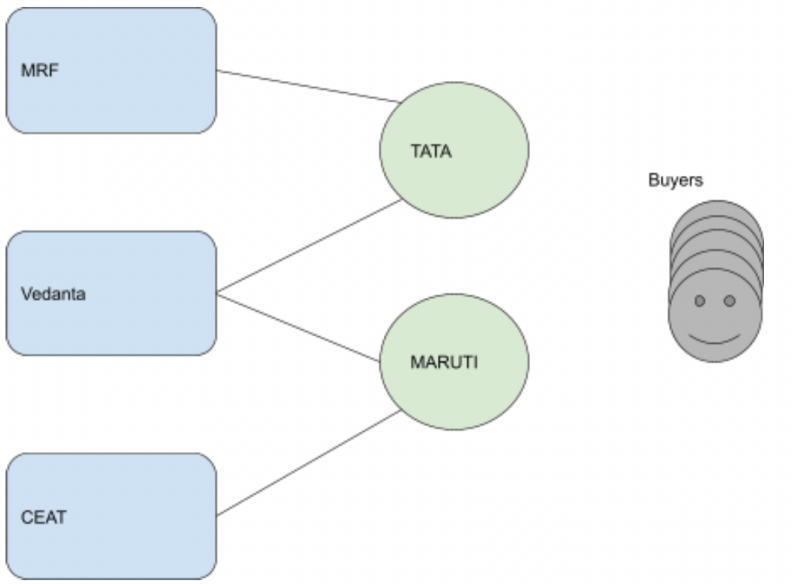
```
SmartContract
├── contracts
        ├── artifacts
        ├──1_SupplyChain.sol
├── scripts
├──tests
        ├──SupplyChain_test.sol
```

Steps to run:

Our SmartContract
Actors:

1. Supplier: There are 3 suppliers (Vedanta, MRF and CEAT). MRF and CEAT manufacture tyres. Vedanta supplies body parts.
2. Manufacturers: There are 2 clients (Tata and Maruti) which procure wheels and tyres
3. Customers/Buyers: Buy products from manufacturers.

Interaction:



Functions:
Our SmartContract handles functions to incorporate the functionalities of all the actors in the smart contract.

1.Supplier:

- addSupplier() :
  Adds supplier to a new struct “supplier” and puts it in a map “suppliers”. An array “supplier_names” keeps track of the supplier name.

- set_limit():
  Supplier **sets a limit on the supply** that can be supplied to be used by manufacturers.
  Each material(a struct containing the details of brand and the type - tyre or body) has a unique materialID set to it, so that there is no duplication of material.

- get_limit():
  The limit of the supplier is returned.

2. Manufacturer:

- addManufacturer() :
  Adds manufacturer to a new struct “manufacturer” and puts it in a map “manufacturers”. An array “manufacturer_names” keeps track of the manufacturer name.

- set_bids():
  Manufacturer **places bids** to the supplier.

- make_product():
  This function is used to **create a new product** and set its pid, brand name, type and other details. Everytime a product is made, the number of tyres, body reduces by one and the product increases. We can delete these ids from our mappings. This guarantees a unique product as product id can only increment and never decrement.

- set_price():
  Manufacturer **sets the price** and defines the retail amount on which he will sell the product to the buyer.

3. Buyer

- buy_product():
  The buyer can **buy a product from manufacturer** using pid and verify if the product exists and if it is authentic, both by material and product supply, by supplier and manufacturer.

- verify():
  Function to check if the product is valid, if the materials used are valid or not. We first check if the address belongs to the buyer, If true, he can check validity of the product.
  If the product is valid, it will have an address which is not the default 0 address(if it doesn't exist). If it is valid, uid exists so the manufacturer who made the product exists.
  We then check if the products with which it was made exists too. We see they all have unique ids which are non-zero and thus they exist and are unique. We also see if the product is not a duplicate by using the property that our uids are unique and the manufacturer's uid is not the same as owner who is now the buyer.

4.Smart Contract

- resource_allocator():
  The algorithm is declared internal. It performs **Resource Optimal Distribution** of materials inside the contract itself. on set_bids(), this function gets called.
  It uses keccak256 to hash the supplier name as only a hash of strings can be compared, to find the supplier names.
  If the supplier is MRF or CEAT, they produce tyre supply. So we check the length of the bid and grant the request if it is 1 and we can satisfy the request. If the request is more than the available limit, we only grant the limit available by the supplier.
  If the supplier is Vedanta,we check if 2 bids are present - given by both manufacturers. We check fairness of bids by seeing if both competing bids have procured tires or not.
  If we can satisfy both bids, supplier does so. This is case 1.
  Else if only enough material exists to satisfy one manufacturer, we pick the manufacturer who has higher price per item and sufficient number of tyres to make the product.
  After that, we reduce our supply limit, and update that in our mapping.
- sendmoney()
  The contract can send money to accounts internally.

Goals:
Goal 1: Bidding is kept secret as no one can access the bids. It is **internal only to the smart contract**. Only resource_allocator() function can open the values and check prices and quantity. This is abstracted even from manufacturers and suppliers.

Goal 2: Optimal Supply of material is performed by the resource allocation algorithm we have defined in our resource_allocator() function. Thus it is a decentralised and trustless supply-chain.

Goal 3: Validity of product is ensured by using unique uids for products and the individual materials. They are only **incremental** in nature. They cannot be decremented. The blockchain encrypts the smart contract and uses hash pointers at each stage to hash our values, so no supplication or manipulation can occur. For the same reason, double spending cannot happen as buyers can perform verification by checking the uids and pids and matching with the address of the supplier/manufacturer which we store in our mapping. These addresses are unique and cannot be duplicated. These are intrinsic properties of the blockchain, as a smart contract may use currencies as non-fungible tokens but it in itself remains unique because of the hashing properties and the way the blockchain is built.